Objektorientert programmering i Python: Introduksjon

IN1000
Høst 2018 – uke 7
Siri Moe Jensen

Læringsmål uke 7

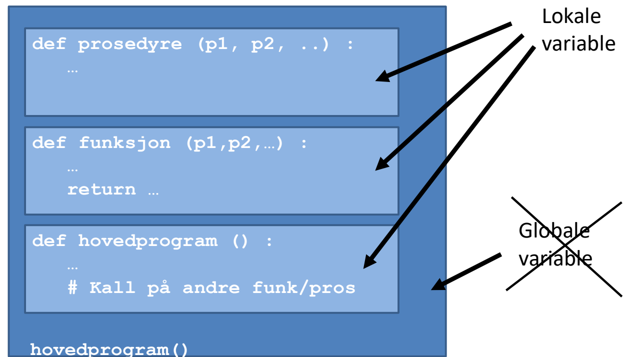
- Kjenne til motivasjon og bakgrunn for objektorientert programmering
- Kunne definere en klasse, opprette og bruke objekter
- Forstå sentrale begreper som grensesnitt og innkapsling
- Kjenne til utviklingsprosessen for en klasse gjennom design, implementasjon og testing

Innhold uke 7

Lite tilbakeblikk:
- Prosedyrer og funksjoner

Objektorientert programmering
- Introduksjon: Hvorfor, hva & hvordan i IN1000
- Programmering med objekter
- Å lage en ny type objekter: Skriv en klasse!
- Utviklingsprosess for klasser

Lite tilbakeblikk: Programflyt og skop



Lite tilbakeblikk: Funksjoner er uttrykk

- Prosedyre tar 0-n argumenter – returnerer ingen verdi
- Funksjoner tar 0-n argumenter – returnerer én verdi

- Når vi trenger en verdi i et program har vi flere muligheter:
  - er det alltid den samme? Literal/ konstant
  - Har vi lagret verdien i en variabel?
  - Kan vi regne den ut?
  - Kan vi finne den ved å kalle på en funksjon?

Uttrykk

Lite tilbakeblikk: Funksjoner er uttrykk

- Funksjoner kan kombineres:

```
alder = int(input ("Din alder: "))
```

- funksjonen `input` returnerer en verdi som brukes direkte som argument til funksjonen `int`.
- `int` returnerer en verdi som legges i variabelen `alder`.

Bakgrunn

- Store og komplekse systemer etter ~20 år programmering med (noen av) de mekanismene dere har lært hittil.
- Hovedsakelig matematiske beregninger – komplekse, men formelt definerte - store tallmengder, men repetitive operasjoner.
- Kompleksitet ga høye kostnader og feil. Arbeidsdeling og gjenbruk ble vanskelig.

Hva skjedde ~1967?

Samtidig:

- Datamaskinene ble kraftigere
- billigere
- mer utbredt

=> Nye anvendelser!

Forsvaret?

Trafikk-planlegging?

Pasientjournal-systemer?

Facebook??

Filosofisk: Konseptet objektorientering

- Å programmere er å modellere
  - Vi bygger en egen representasjon av fenomener (konkrete eller abstrakte) vi trenger å manipulere i datamaskinen
  - denne representasjonen er ikke virkeligheten – men kan holde rede på noen aspekter som er viktige for det vi skal lage et program for
- Vi lager en *modell av "virkeligheten"*
- Å programmere er å forstå
  - virkeligheten
  - det problemet/ behovet vi skal løse
  - hva som kan/ vil endre seg i fremtiden

hvilke data er viktige? Hvordan henger de sammen? hva må jeg gjøre med dem? Hvilke sammenhenger trenger jeg å ta hensyn til? hvilke ferdige programmer kan jeg bruke? skal noen bruke min kode til andre ting?

Mer konkret: En måte å strukturere programmer på

- *Prosedyrer og funksjoner* kan brukes for å løse deloppgaver
- Hva om deloppgavene handler om å bearbeide felles, kanskje komplekse, data?

Forslag:

- Samle relaterte data og kode for å manipulere dem i **objekter**
- Objektene tilbyr resten av programmet et sett **metoder** grensesnitt
- Hvordan objektet representerer og manipulerer sine data er skjult innkapsling

Et objekt representerer et "fenomen" vi ønsker å manipulere

Vi har tidligere brukt objekter som

- ..representerer lister m/ tilhørende metoder

```
navneliste = []
navneliste.append("Per")
navneliste.append("Paal")
navneliste.append("Espen")
navn = navneliste.pop()
```

- ..representerer filer m/ tilhørende metoder

```
utfil = open ("navn.txt", "w")
utfil.write (navn)
utfil.close()
```

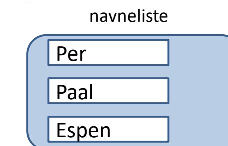
`List`-objekter

```
navneliste = []
```

- Høyresiden oppretter et objekt – en liste som foreløpig er tom – og legger objektet i variabelen i venstresiden

```
navneliste.append("Per")
navneliste.append("Paal")
navneliste.append("Espen")
navn = navneliste.pop()
```

navneliste



- Vi kan utføre listemetoder på objektet vårt ved hjelp av kallet *< variabelnavn.metodenavn(parametere) >*

Metoder

- Vi kaller på metoder omtrent som prosedyrer og funksjoner
- MEN en metode alltid hører til et objekt, derfor bruker vi "dot-notasjon"

```
minListe.append(nyttElem)
```

- En metode kan returnere en verdi (som funksjon)

Kall på metode

- uten returverdi

```
navneliste = []

navneliste.append("Per")
navneliste.append("Paal")
```

- med returverdi

```
navneliste = []

navneliste.append("Per")
navneliste.append("Paal")
navn = navneliste.pop()
```

Grensesnitt

- Et objekt tilbyr ett sett metoder som programmerere kan bruke for å lese av, endre eller aktivisere et objekt: Objektets grensesnitt
- Grensesnittet bestemmes av *objektets klasse*
- Hvordan finner vi ut hvilke metoder som tilbys av objekter av en bestemt klasse?
- F eks i læreboka eller Python dokumentasjon
  - google search: python methods List
  - f eks *https://docs.python.org/3/tutorial/datastructures.html*
  - *(nb bl.a. Python versjoner)*

Klasser

- En klasse er et mønster/ oppskrift for objekter av samme type
- String, List og File er eksempler på innebygde Python-klasser med hver sine grensesnitt
- Du kan definere (programmere) dine egne klasser som du så kan opprette objekter av

Egne klasser

- Eks:
  - klasse **Student** med data om en student og verktøy for å arbeide med denne
  - klasse **Dato** med data for dag, måned, år og verktøy for manipulasjon av disse
- Kan opprette ett eller mange objekter av hver klasse
- Objekter av samme klasse tilbyr samme grensesnitt og har samme data – men typisk med ulike verdier.

Hvorfor lage en klasse?

- For å tilby et sett tjenester som hører sammen (verktøykasse)
- Spesielt nyttig for tjenester som trenger å dele data over tid (eks for håndtering av en liste, eller en fil, eller en tekst)
- For å kunne lage flere like objekter med samme data og oppførsel (eks representere studenter)
- For å simulere sammensatte fenomener vha objekter som representerer virkelige elementer (eks trafikk- simuleringer)

Eksempel Student

- Skal lagre informasjon om studenter
- Studenter har MANGE egenskaper – hva skal jeg bruke klassen til?
  - Holde rede på oppmøte i gruppetimene
  - Identifisere med navn (utvidelse)

Student: Grensesnitt uformelt

- Ønsker å kunne gjøre følgende:
  - Lage et Student-objekt med gitt navn og null oppmøte
  - Øke oppmøte med 1 ved tilstedeværelse
  - Lese av / hente ut oppmøte ved behov

Implementasjon

- Hvordan programmereren koder klassen
- "Usynlig" (privat) for programmerere som bruker klassen til å lage objekter
- Klassedefinisjonen bestemmer
  - hvilke instansvariabler skal klassens objekter ha (dataene)
  - hvordan utføres metodene (operasjonene)
- Hvert objekt får et eget sett instansvariabler som initialiseres ved opprettelsen
- Instansvariablene er private: Brukes kun av klassens metoder

Hvordan lage egne klasser?

```
class Student :

  def __init__(self) :
    self._antMott = 0

  def registrer(self) :
    self._antMott = self._antMott + 1

  def hentOppmote(self) :
    return self._antMott
```

klassenavn

konstruktør

metode

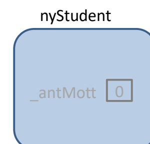
funksjons-metode

..og hvordan opprette et objekt?

- Noen innebygde klasser har snarveier for opprettelse av objekter
  - "" oppretter en ny **str**
  - [] oppretter en ny **list**
- Ellers opprettes objekter slik:

```
nyStudent = Student()
```

nyStudent



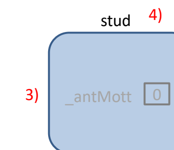
Hvert objekt får sine egne data

- **Instansvariable** opprettes og initialiseres i __init__ metoden (konstruktøren)
- Konstruktøren kalles automatisk når et nytt objekt opprettes:
  - oppretter objektet
  - oppretter instansvariablene i objektet
  - initierer objektvariablene
  - returnerer det nye objektet til kallstedet

```
class Student :
2) def __init__(self) :
3)   self._antMott = 0

stud = Student() 1)
4)
```

stud 4)

3)



Student: Grensesnitt i Python

```
## Behandler en students oppmote på gruppetime
#
class Student :
  ## Oppretter student med gitt navn og oppmote 0
  #
  def __init__(self, studNavn) :

  ## Registrer oppmote
  #
  def registrer(self) :

  ## Hent ut oppmote
  #
  def hentOppmote(self) :
```

Student: Implementasjon i Python

```
## Antall oppmøter på gruppetime for en student
#
class Student :
  ## Oppretter student med gitt navn og oppmøte 0
  #
  def __init__(self, studnavn) :
    self._navn = studnavn
    self._antMott = 0

  ## Registrer oppmote
  #
  def registrer(self) :
    self._antMott = self._antMott + 1
```

Student: Implementasjon i Python

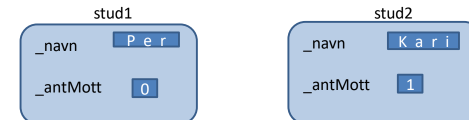
forts

```
## Antall oppmøter på gruppetime for en student
#
class Student :


# …fortsetter fra forrige slide…

  ## Hent ut oppmote
  #
  def hentOppmote(self) :
    return self._antMott
```

Studentobjekter

```
stud1 = Student("Per")
stud2 = Student("Kari")
stud2.registrer()

print(stud1.hentOppmote())
print(stud2.hentOppmote())
```

```
>python testStudent.py
0
1
>
```

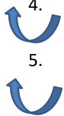


⇒ OPPGAVE (2 eller 3 sammen):
- Hvilken metode mangler vi for at utskriften skal være forståelig?
- Utvid klassen Student, diskuter gjerne alternativer

Fremgangsmåte: OOP

1. Identifiser aktuelle klasser (hva er sentrale «ting»/ begreper programmet skal behandle?)

2. Design klassens grensesnitt (hvilke operasjoner trenger jeg for objekter av klassen?)
3. Design klassens datarepresentasjon (hvordan skal objektene representere dataene sine?)
4. Implementer (fyll ut) metodene (skriv klassen ferdig)



5. Lag et testprogram som oppretter et eller flere objekter og kaller på metodene i objektenes grensesnitt

Test av klassen

- Vi bruker Pythons (interaktive) shell eller skriver et testprogram i en egen fil
- Importerer klassen Student
- Oppretter (minst) et objekt
- Kaller på alle metoder (minst) en gang - ulike sekvenser kan være nødvendig for å avdekke feil siden objekter husker verdiene sine (har en tilstand)
- `assert` tvinger oss til å tenke ut på forhånd hva vi forventer

Innkapsling

- Alle endringer av et objekts instansvariable skal skje ved hjelp av klassens metoder (grensesnitt/ public interface)
- Metodene inne i klassen kan endre og lese av variabelen **`_antMott`** – mens kode utenfor klassen må kalle en metode på et objekt for å lese den av eller endre den i objektet.
- Innkapsling er et sentralt oo prinsipp
  - kode kan endres så lenge grensesnittet er det samme
  - gjennomført ansvarsdeling (splitt og hersk)

Eksempel II: Utgiftsregnskap

- Hvor mye penger har jeg brukt (f eks ila en dag på reise, varer i handlevognen, eller en kveld ute)
- Registrerer fortløpende
- Leser av ved behov
- Vi skal programmere en klasse som hjelper oss med dette

Utgifter: Uformelt grensesnitt

- Klassen Utgifter holder rede på det vi bruker
- Hvilke operasjoner skal tilbys?
  - Registrer en ny utgift, dvs øke summen av hittil registrerte beløp
  - Lese av summen av hittil registrerte beløp
- Dette beskriver **grensesnittet** for Utgifter-klassen med ord

Utgifter: Grensesnitt i Python

- En metode for hver operasjon

```
# Legger til en ny utgift
def leggTil(self, utgift) :


# returnerer totalbelop brukt hittil
def brukt(self) :
```

- Dette er klassen grensesnitt uttrykt i Python
- Et komplett grensesnitt skal senere inkludere standardisert dokumentasjon av hver metode

Oppgave:
Implementer klassen Utgifter med dette grensesnittet

Utgifter: Implementasjon

```
class Utgifter :
  # trenger en belop-variabel, initialiserer med 0
  def __init__(self) :
    self._belop = 0.0

  # metode som legger til en ny utgift
    def leggTil(self, utgift) :
      self._belop = self._belop + utgift

  # returnerer totalbelop brukt hittil
    def brukt(self) :
       return self._belop
```

Program med prosedyrer og funksjoner

mittProgram.py

```
def prosedyre (p1, p2, ..) :
   …

def funksjon (p1,p2,…) :
   …
   return …

def hovedprogram () :
   …
   # Kall på andre funk/pros

hovedprogram()
```

Program med klasser

mittKlasseProgram.py

```
class Klasse1 :
   def __init__(self,p1,…) :
      …
      self._instVar = …

   def metode1 (self, p2,..) :
      …

def hovedprogram () :
   …
   # oppretter og bruker objekter

hovedprogram()
```

Oppsummering uke 7

Flere perspektiver på objektorientert programmering:

- kodeperspektiv; en måte å strukturere programmene på/ arkitektur
- historisk/ utviklingsperspektiv; støtter (sikker) implementasjon av større systemer i takt med teknologiutviklingen
- modellperspektiv; hvordan beskrive, forstå og dele mer komplekse fenomener mellom flere interessenter

Utviklingsprosess for et objektorientert program:

Klasse(r) - grensesnittet - innhold - teste

Grensesnitt: Et sett dokumenterte metoder som tilbys av en klasse

Innkapsling: Skjule implementasjon av klassen for de som bruker den